CS3723 Pgm Assignment #1: Storage Management in C (50 pts) – Due Monday Sept 24th, 2018 at 11:59pm.

In this assignment, you will create **heap storage management functions** based on using reference counts and understanding metadata. You will write functions which manage free memory lists and the reference counts for user data nodes:

- When a node is created, it sets the count to 1 (this is like a variable referencing data causing the count to be 1).
- A node (i.e., *from node*) can be associated with another node (i.e., *to node*) by changing a pointer attribute in the *from* node. This will cause the *to node* to have its reference count increased. If the *from node* was already referencing another node via that particular pointer attribute, that original referenced node should have its reference count decreased. If the new *to node* value is NULL, don't attempt to increase its reference count.
- We can also add references to nodes similar to a node having another variable referencing it.

Your code must be able to understand the metadata for any nodes. If you hard-code your program to only handle the two referenced node types, you will lose at least 80% of the points. To better understand the metadata, please examine the sample partial output below.

It is likely that a question on the midterm exam will be easier if you do this programming assignment. Total points: 50 (for this assignment) + (apprx) 20 * 3 = (apprx) 110 pts

To help reduce the code that you have to write and reduce debugging difficulty, I have provided a **driver program** which has many capabilities. Initially, it will set up metadata and the heap. It also has functions to print the metadata, process commands from an input file, print the contents of user nodes, and print the heap. The driver program also provides multiple examples of pointer manipulation.

Files that I provided (located in **/usr/local/courses/clark/cs3723/2018Fa)**:

cs3723p1Driver.c - reads the input file, calls your reference count storage management functions, and uses a hash table to store the addresses of allocated memory (so that those can be subsequently associated with other nodes or freed). An important function is the **setData** function which uses metadata to set attributes in a node. The driver also provides functions for printing the heap memory to help with debugging.

cs3723p1.h - include file for this program. Some important typedefs:

MetaAttr – describes one attribute in a node type: name, type, size in bytes, and offset

NodeType – describes one node type: name, beginning subscript in metaAttrM array, total size

AllocNode - contains the node's size, allocated/free flag, reference count, node type, and the user's data.

FreeNode – used for nodes that have been freed (and not yet reallocated). It contains a pointer to the next free node in a linked list of free nodes.

StorageManager - a structure that contains the address of the heap (pBeginStorage), free memory pointer, an array of pointers to free lists (one for each node type), an array of NodeType entries, and an array of MetaAttr entries. It does not have a count of the number of entries in those arrays. Instead, sentinels are used to mark the end of the arrays.

SMResult - used by many of the storage management functions to specify whether they executed successfully.

hashApi.cpp - C++ code to integrate C with the C++ Hash Table Class (unordered_map). This supports functions getHash, putHash, eraseAll, and printAll. This is only used by the driver.

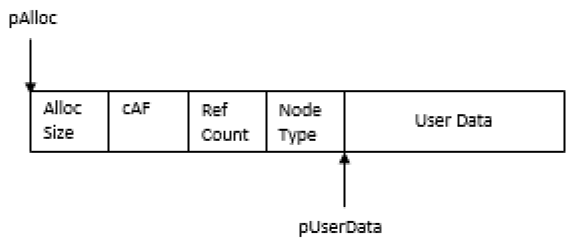
p1Input.txt - Input text file suitable for the driver. The driver uses **stdin** so redirect input from this file.

printNode.o - object code for printing a node.

makefile - Please use this makefile to create your **p1** executable. Note that you should not use the hen servers. The makefile uses **g++** instead of gcc. To build the executable (with it automatically building the other pieces), type

```
make p1
```

In C, **malloc**() returns a pointer to *user data* and we pass **free**() a pointer to *user data.* As you know from lecture, **malloc**() actually allocated more bytes than you requested, including an attribute for the total size. The following diagram shows that our allocated nodes(for this assignment) contain an allocated size,allocated/free flag (cAF), reference count, and node type:



When integrating with a **user** of the storage management functions, the storage management software uses pUserData (a pointer to the user data). The function **userAllocate** allocates a node, but returns a pointer to the user data portion of the node. Similarly, the other user integration functions (names begin with "user") are passed pointers to user data instead of a pointer to the beginning of the allocated node.

You will need to code the following functions; however, due to modularity concerns, you may want to create extra functions. Your code should be placed in cs3723p1.c.

```
void * userAllocate(StorageManager *pMgr, short shUserDataSize, short shNodeType
    , char sbUserData[], SMResult *psmResult)
```

- Purpose: allocates an AllocNode from our heap, copies the binary sbUserData into that node, and returns a pointer to the user data portion of the node.
- It is passed:
  - pMgr – a pointer to the StorageManager structure. (See cs3723p1.h for more information.)
  - shUserDataSize – the size of the user data. This does not include the four storage management prefix attributes. The size of the allocated node will actually be this plus at least the pMgr->shPrefixSize.
  - shNodeType – this is a subscript into the storage manager's nodeTypeM array.
  - sbUserData – this is storage buffer containing the actual user data which needs to be placed in the allocated node. This data can contain integer and double values; therefore, it is not a zero-terminated string.
  - psmResult – a pointer to a SMResult structure. (See cs3723p1.h for more information.)
- This function first checks to see if there is a node type-specific free node available in pMgr -> pFreeHeadM[shNodeType]. If there is, it is used and the free list is updated. If there is not, it uses the driver-provided **utilAllocate** to allocate memory from the top of the heap. Note that your code will **not** use malloc().
- Initializes an AllocNode:
  - Set its reference count to 1.
  - Set its node type.
  - Set its allocated size which is bigger than shUserDataSize.
  - Sets its cAF to indicate that it is allocated.
  - Set its sbData. Why can't you use strcpy for this?
- Sets the psmResult information.
- Returns a pointer (from the user's perspective) to the allocated memory (i.e., a pointer to where the user data is in the node). This is not the address of the AllocNode! (See the diagram.) If memory was not allocated, it should return NULL.

```
void userRemoveRef(StorageManager *pMgr, void *pUserData, SMResult *psmResult)
```

- Purpose: removes a reference to the specified data.
- It is passed:

    - pMgr – a pointer to the StorageManager structure. (See cs3723p1.h for more information.)
    - pUserData – a pointer to the user data within an allocated node.
    - psmResult – a pointer to a SMResult structure. (See cs3723p1.h for more information.)
- Decrements the reference count for the corresponding AllocNode.
- If the **reference count reaches zero**, it must "free" the AllocNode:
    - If the user node type references pointers, decrement the ref count on any directly referenced user data nodes*. (*This will be a recursive call to userRemoveRef*.)*
    - Frees the AllocNode which reached a ref count of 0 by calling **memFree**. Note that memFree keeps the "freed" node around . It does not do a C free().

```
void userAssoc(StorageManager *pMgr, void *pUserDataFrom, char szAttrName[]
    , void *pUserDataTo, SMResult *psmResult)
```

- Purpose: logically, it is setting a pointer in a user node to point to another node (or NULL) and correspondingly updating reference counts.
- It is passed:
    - pMgr – a pointer to the StorageManager structure. (See cs3723p1.h for more information.)
    - pUserDataFrom – a user data pointer to the *from node* which contains the pointer attribute. This is the *from node.*
    - szAttrName – the name of the attribute which is used to determine where the pointer is located within the *from node.*
    - pUserDataTo – a user data pointer to the *to node.*
    - psmResult – a pointer to a SMResult structure. (See cs3723p1.h for more information.)
- The referenced attribute must be a pointer. If not, set psmResult->rc to `RC_ASSOC_ATTR_NOT_PTR`.
- If that specified pointer attribute in the *from node* is already referencing something, that referenced user data node needs to have its reference count decremented (how should you do that? could it reach 0?)
- Unless it is NULL, the new referenced user data node needs to have its ref count increased.
- Change the user pointer in the specified user data node to point to the new referenced user data node or NULL (if that was specified).
- Set the psmResult information.

```
void userAddRef(StorageManager *pMgr, void *pUserDataTo, SMResult *psmResult)
```

- Purpose: logically, it is adding a reference to the specified *to node.* This is similar to adding a variable's reference to the *to node*.
- It is passed:
    - pMgr – a pointer to the StorageManager structure. (See cs3723p1.h for more information.)
    - pUserDataTo – a user data pointer to the *to node.*
    - psmResult – a pointer to a SMResult structure. (See cs3723p1.h for more information.)
- Increase the reference count for the *to node* by one.

```
void memFree(StorageManager *pMgr, AllocNode *pAlloc, SMResult *psmResult)
```

- Purpose: This is adding the specified allocated node to the free list for that node's node type.
- It is passed:
    - pMgr – a pointer to the StorageManager structure. (See cs3723p1.h for more information.)
    - pAlloc – a pointer to an allocated node. This is NOT a user data pointer.
    - psmResult – a pointer to a SMResult structure. (See cs3723p1.h for more information.)
- Verify that the node is in fact an allocated node. If not return an error via psmResult.
- Add this node to the **front** of the free list for this node's node type.

Notes:

1. In this program, **your code must not** use **malloc**(), **calloc**() or **free**().
2. To help understand how to use the metadata, examine the driver's **setData** function.
3. Your code must follow my **programming standards**.
4. You must make certain your code works on a fox server and can be compiled by the specified makefile.

5. To simplify grading, please include turn in a zip file (named LastnameFirstname.zip). It should contain:
   - cs3723p1.c – your C source code
   - p1Output.txt – your output
6. For Microsoft Visual Studio Users:
   - If you need a wider Console Window:
     - Once the console window displays (you may want a break point in your code so that it doesn't disappear), click the top left corner of the console window.
     - Properties
     - Layout
     - Change the Screen Buffer Size to 120
     - Change the Window Size to 120
   - Unfortunately, you will have to create your own printNode.
7. Make certain your code runs correctly on a **fox** server.
8. Via BlackBoard, turn in a zip file (named *LastnameFirstname*.zip) which contains:
   - Your **cs3723p1.c** source file
   - Your output named **p1Output.txt**

Sample Partial Output:

```
Metadata
Node Type  Beg Attr Sub Total Sz
Customer      0           56
               Attribute Name Type Offset Size
               customerId       S       0   12
               name             S      12   20
               pFirstItem       P      32    8
               pNextCust        P      40    8
               balance          D      48    8
LineItem      5           32
               Attribute Name Type Offset Size
               productId        S       0   10
               iQtyReq          I      12    4
               dCost            D      16    8
               pNextItem        P      24    8
>>> ALLOC C111 Customer 111,Sal A Mander,NULL,NULL,100.00
>>> PRTNODE C111
       Alloc Address  Size NodeType  RefCnt  DataAddress
       0x1def070        64      0       1     0x1def078
                      Attr Name      Type Value
                      customerId       S  111
                      name             S  Sal A Mander
                      pFirstItem       P  (nil)
                      pNextCust        P  (nil)
                      balance          D  100.000000
>>> ALLOC C222 Customer 222,Barb Wire,NULL,NULL,200.00
>>> PRTNODE C222
       Alloc Address  Size NodeType  RefCnt  DataAddress
       0x1def0b0        64      0       1     0x1def0b8
                      Attr Name      Type Value
                      customerId       S  222
                      name             S  Barb Wire
                      pFirstItem       P  (nil)
                      pNextCust        P  (nil)
                      balance          D  200.000000
>>> ALLOC PPF001 LineItem PPF001,5,9.95,NULL
>>> PRTNODE PPF001
       Alloc Address  Size NodeType  RefCnt  DataAddress
       0x1def0f0        40      1       1     0x1def0f8
                      Attr Name      Type Value
                      productId        S  PPF001
                      iQtyReq          I  5
                      dCost            D  9.950000
                      pNextItem        P  (nil)
*
* #1 associate customer 111 with a next pointing to 222
*
>>> ASSOC C111 pNextCust C222
* customer 111's ref cnt should still be 1, but its pNextCust should point to 222
>>> PRTNODE C111
       Alloc Address  Size NodeType  RefCnt  DataAddress
       0x1def070        64      0       1     0x1def078
                      Attr Name      Type Value
                      customerId       S  111
                      name             S  Sal A Mander
                      pFirstItem       P  (nil)
                      pNextCust        P  0x1def0b8
                      balance          D  100.000000
* customer 222's ref cnt should now be 2
>>> PRTNODE C222
       Alloc Address  Size NodeType  RefCnt  DataAddress
       0x1def0b0        64      0       2     0x1def0b8
                      Attr Name      Type Value
                      customerId       S  222
                      name             S  Barb Wire
                      pFirstItem       P  (nil)
                      pNextCust        P  (nil)
                      balance          D  200.000000
*
* associate customer 111 to PPF001
*
>>> ASSOC C111 pFirstItem PPF001
>>> PRTNODE C111
       Alloc Address  Size NodeType  RefCnt  DataAddress
       0x1def070        64      0       1     0x1def078
                      Attr Name      Type Value
                      customerId       S  111
                      name             S  Sal A Mander
                      pFirstItem       P  0x1def0f8
                      pNextCust        P  0x1def0b8
                      balance          D  100.000000
* associate customer 222 to 333
>>> ALLOC C333 Customer 333,Misty Wind,NULL,NULL,70.00
>>> ASSOC C222 pNextCust C333
*
* #2 111 should point to 222 which points to 333
*    111 should also point to PPF001
*
>>> PRTALL
PPF001:0x1def0f8
       Alloc Address  Size NodeType  RefCnt  DataAddress
       0x1def0f0        40      1       2     0x1def0f8
                      Attr Name      Type Value
                      productId        S  PPF001
                      iQtyReq          I  5
                      dCost            D  9.950000
                      pNextItem        P  (nil)
C333:0x1def120
       Alloc Address  Size NodeType  RefCnt  DataAddress
       0x1def118        64      0       2     0x1def120
                      Attr Name      Type Value
                      customerId       S  333
                      name             S  Misty Wind
                      pFirstItem       P  (nil)
                      pNextCust        P  (nil)
                      balance          D  70.000000
C222:0x1def0b8
       Alloc Address  Size NodeType  RefCnt  DataAddress
       0x1def0b0        64      0       2     0x1def0b8
                      Attr Name      Type Value
                      customerId       S  222
                      name              S  Barb Wire
                      pFirstItem        P  (nil)
                      pNextCust         P  0x1def120
                      balance           D  200.000000
C111:0x1def078
       Alloc Address  Size NodeType  RefCnt  DataAddress
       0x1def070         64      0       1     0x1def078
                      Attr Name      Type Value
                      customerId        S  111
                      name              S  Sal A Mander
                      pFirstItem        P  0x1def0f8
                      pNextCust         P  0x1def0b8
                      balance           D  100.000000

* add another customer
>>> ALLOC C444 Customer 444,Emory Board,NULL,NULL,44.44
>>> ADDREF PC444 C444
* #3 Customer 444 should have a ref count of 2
>>> PRTNODE C444
       Alloc Address  Size NodeType  RefCnt  DataAddress
       0x1def158         64      0       2     0x1def160
                      Attr Name      Type Value
                      customerId        S  444
                      name              S  Emory Board
                      pFirstItem        P  (nil)
                      pNextCust         P  (nil)
                      balance           D  44.440000
```